

100- Rom
attached
EV316936954

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

APPLICATION FOR LETTERS PATENT

Programming Interface for a Computer Platform

Inventor(s):

Rob Relyea

Jeff Bogdan

ATTORNEY'S DOCKET NO. MS1-1780US

TECHNICAL FIELD

This invention relates to software and to development of such software. More particularly, this invention relates to a programming interface that facilitates use of a software platform by application programs and computer hardware.

BRIEF DESCRIPTION OF ACCOMPANYING COMPACT DISCS

Accompanying this specification is a set of three compact discs that stores a Software Development Kit (SDK) for the Microsoft® Windows® Code-Named "Longhorn" operating system. The SDK contains documentation for the Microsoft® Windows® Code-Named "Longhorn" operating system. Duplicate copies of each of these three compact discs also accompany this specification.

The first compact disc in the set of three compact discs (CD 1 of 3) includes a file folder named "lhsdk" that was created on October 22, 2003; it is 586 Mbytes in size, contains 9,692 sub-folders, and contains 44,292 sub-files. The second compact disc in the set of three compact discs (CD 2 of 3) includes a file folder named "ns" that was created on October 22, 2003; it is 605 Mbytes in size, contains 12,628 sub-folders, and contains 44,934 sub-files. The third compact disc in the set of three compact discs (CD 3 of 3) includes a file folder named "ns" that was created on October 22, 2003; it is 575 Mbytes in size, contains 9,881 sub-folders, and contains 43,630 sub-files. The files on each of these three compact discs can be executed on a Windows®-based computing device (e.g., IBM-PC, or equivalent) that executes a Windows®-brand operating system (e.g., Windows® NT, Windows® 98, Windows® 2000, Windows® XP, etc.). The files on each compact disc in this set of three compact discs are hereby incorporated by reference.

1 Each compact disc in the set of three compact discs itself is a CD-R, and
2 conforms to the ISO 9660 standard. The contents of each compact disc in the set
3 of three compact discs is in compliance with the American Standard Code for
4 Information Interchange (ASCII).

5 6 **BACKGROUND**

7 Very early on, computer software came to be categorized as “operating
8 system” software or “application” software. Broadly speaking, an application is
9 software meant to perform a specific task for the computer user such as solving a
10 mathematical equation or supporting word processing. The operating system is
11 the software that manages and controls the computer hardware. The goal of the
12 operating system is to make the computer resources available to the application
13 programmer while at the same time, hiding the complexity necessary to actually
14 control the hardware.

15 The operating system makes the resources available via functions that are
16 collectively known as the Application Program Interface or API. The term API is
17 also used in reference to a single one of these functions. The functions are often
18 grouped in terms of what resource or service they provide to the application
19 programmer. Application software requests resources by calling individual API
20 functions. API functions also serve as the means by which messages and
21 information provided by the operating system are relayed back to the application
22 software.

23 In addition to changes in hardware, another factor driving the evolution of
24 operating system software has been the desire to simplify and speed application
25 software development. Application software development can be a daunting task,

1 sometimes requiring years of developer time to create a sophisticated program
2 with millions of lines of code. For a popular operating system such as various
3 versions of the Microsoft Windows® operating system, application software
4 developers write thousands of different applications each year that utilize the
5 operating system. A coherent and usable operating system base is required to
6 support so many diverse application developers.

7 Often, development of application software can be made simpler by making
8 the operating system more complex. That is, if a function may be useful to several
9 different application programs, it may be better to write it once for inclusion in the
10 operating system, than requiring dozens of software developers to write it dozens
11 of times for inclusion in dozens of different applications. In this manner, if the
12 operating system supports a wide range of common functionality required by a
13 number of applications, significant savings in applications software development
14 costs and time can be achieved.

15 Regardless of where the line between operating system and application
16 software is drawn, it is clear that for a useful operating system, the API between
17 the operating system and the computer hardware and application software is as
18 important as efficient internal operation of the operating system itself.

19 When developing applications, developers use a variety of tools to generate
20 graphical items and other content. Additional tools are available to arrange
21 graphical items and other data to be displayed or rendered. These tools are
22 typically created by different entities or different tool developers. As a result, the
23 tools do not provide a consistent programming environment. Thus, a developer
24 using these different tools needs to learn how to utilize each of the tools and
25 attempt to make them communicate with one another. These activities can be

1 tedious and time consuming, taking time away from the actual development task at
2 hand.

3 Over the past few years, the universal adoption of the Internet, and
4 networking technology in general, has changed the landscape for computer
5 software developers. Traditionally, software developers focused on single-site
6 software applications for standalone desktop computers, or LAN-based computers
7 that were connected to a limited number of other computers via a local area
8 network (LAN). Such software applications were typically referred to as “shrink
9 wrapped” products because the software was marketed and sold in a shrink-
10 wrapped package. The applications utilized well-defined APIs to access the
11 underlying operating system of the computer.

12 As the Internet evolved and gained widespread acceptance, the industry
13 began to recognize the power of hosting applications at various sites on the World
14 Wide Web (or simply the “Web”). In the networked world, clients from anywhere
15 could submit requests to server-based applications hosted at diverse locations and
16 receive responses back in fractions of a second. These Web applications, however,
17 were typically developed using the same operating system platform that was
18 originally developed for standalone computing machines or locally networked
19 computers. Unfortunately, in some instances, these applications do not adequately
20 transfer to the distributed computing regime. The underlying platform was simply
21 not constructed with the idea of supporting limitless numbers of interconnected
22 computers.

23 To accommodate the shift to the distributed computing environment being
24 ushered in by the Internet, Microsoft Corporation developed a network software
25 platform known as the “.NET” Framework (read as “Dot Net”). Microsoft® .NET

1 is software for connecting people, information, systems, and devices. The
2 platform allows developers to create Web services that will execute over the
3 Internet. This dynamic shift was accompanied by a set of API functions for
4 Microsoft's .NET™ Framework.

5 As use of the .NET™ Framework has become increasingly common, ways
6 to increase the efficiency and/or performance of the platform have been identified.
7 The inventors have developed a unique set of API functions to allow for such
8 increased efficiency and/or performance.

9 10 **SUMMARY**

11 A programming interface, such as an API, provides functions for generating
12 applications, documents, media presentations and other content. These functions
13 allow developers to obtain services from an operating system, object model
14 service, or other system or service. In one embodiment, the functions allow a
15 developer to generate a graphical user interface.

16 17 **BRIEF DESCRIPTION OF THE DRAWINGS**

18 The same numbers are used throughout the drawings to reference like
19 features.

20 Fig. 1 illustrates a network architecture in which clients access Web
21 services over the Internet using conventional protocols.

22 Fig. 2 is a block diagram of a software architecture for a network platform,
23 which includes an application program interface (API).

24 Fig. 3 is a block diagram of the presentation subsystem supported by the
25 API, as well as function classes of the various API functions.

1 Fig. 4 is a block diagram of an exemplary computer that may execute all or
2 part of the software architecture.

3 Figs. 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15 and 16 illustrate various example
4 implementations of a programming interface.

5 6 **DETAILED DESCRIPTION**

7 This disclosure addresses an application program interface (API) for a
8 network platform upon which developers can build Web applications and services.
9 More particularly, an exemplary API is described for operating systems that make
10 use of a network platform, such as the .NET™ Framework created by Microsoft
11 Corporation. The .NET™ Framework is a software platform for Web services and
12 Web applications implemented in the distributed computing environment. It
13 represents the next generation of Internet computing, using open communication
14 standards to communicate among loosely coupled Web services that are
15 collaborating to perform a particular task.

16 In the described implementation, the network platform utilizes XML
17 (extensible markup language), an open standard for describing data. XML is
18 managed by the World Wide Web Consortium (W3C). XML is used for defining
19 data elements on a Web page and business-to-business documents. XML uses a
20 similar tag structure as HTML; however, whereas HTML defines how elements
21 are displayed, XML defines what those elements contain. HTML uses predefined
22 tags, but XML allows tags to be defined by the developer of the page. Thus,
23 virtually any data items can be identified, allowing Web pages to function like
24 database records. Through the use of XML and other open protocols, such as
25 Simple Object Access Protocol (SOAP), the network platform allows integration

1 of a wide range of services that can be tailored to the needs of the user. Although
2 the embodiments described herein are described in conjunction with XML and
3 other open standards, such are not required for the operation of the claimed
4 invention. Other equally viable technologies will suffice to implement the
5 inventions described herein.

6 As used herein, the phrase application program interface or API includes
7 traditional interfaces that employ method or function calls, as well as remote calls
8 (e.g., a proxy, stub relationship) and SOAP/XML invocations.

9 It should be appreciated that in some of namespace descriptions below,
10 descriptions of certain classes, interfaces, enumerations and delegates are left
11 blank. More complete descriptions of these classes, interfaces, enumerations and
12 delegates can be found in the subject matter of the compact discs that store the
13 SDK referenced above.

14 15 EXEMPLARY NETWORK ENVIRONMENT

16 Fig. 1 shows a network environment 100 in which a network platform, such
17 as the .NET™ Framework, may be implemented. The network environment 100
18 includes representative Web services 102(1), ..., 102(N), which provide services
19 that can be accessed over a network 104 (e.g., Internet). The Web services,
20 referenced generally as number 102, are programmable application components
21 that are reusable and interact programmatically over the network 104, typically
22 through industry standard Web protocols, such as XML, SOAP, WAP (wireless
23 application protocol), HTTP (hypertext transport protocol), and SMTP (simple
24 mail transfer protocol) although other means of interacting with the Web services
25 over the network may also be used, such as Remote Procedure Call (RPC) or

1 object broker type technology. A Web service can be self-describing and is often
2 defined in terms of formats and ordering of messages.

3 Web services 102 are accessible directly by other services (as represented
4 by communication link 106) or a software application, such as Web application
5 110 (as represented by communication links 112 and 114). Each Web service 102
6 is illustrated as including one or more servers that execute software to handle
7 requests for particular services. Such services often maintain databases that store
8 information to be served back to requesters. Web services may be configured to
9 perform any one of a variety of different services. Examples of Web services
10 include login verification, notification, database storage, stock quoting, location
11 directories, mapping, music, electronic wallet, calendar/scheduler, telephone
12 listings, news and information, games, ticketing, and so on. The Web services can
13 be combined with each other and with other applications to build intelligent
14 interactive experiences.

15 The network environment 100 also includes representative client devices
16 120(1), 120(2), 120(3), 120(4), ..., 120(M) that utilize the Web services 102 (as
17 represented by communication link 122) and/or the Web application 110 (as
18 represented by communication links 124, 126, and 128). The clients may
19 communicate with one another using standard protocols as well, as represented by
20 an exemplary XML link 130 between clients 120(3) and 120(4).

21 The client devices, referenced generally as number 120, can be
22 implemented many different ways. Examples of possible client implementations
23 include, without limitation, portable computers, stationary computers, tablet PCs,
24 televisions/set-top boxes, wireless communication devices, personal digital
25 assistants, gaming consoles, printers, photocopiers, and other smart devices.

1 The Web application 110 is an application designed to run on the network
2 platform and may utilize the Web services 102 when handling and servicing
3 requests from clients 120. The Web application 110 is composed of one or more
4 software applications 130 that run atop a programming framework 132, which are
5 executing on one or more servers 134 or other computer systems. Note that a
6 portion of Web application 110 may actually reside on one or more of clients 120.
7 Alternatively, Web application 110 may coordinate with other software on clients
8 120 to actually accomplish its tasks.

9 The programming framework 132 is the structure that supports the
10 applications and services developed by application developers. It permits multi-
11 language development and seamless integration by supporting multiple languages.
12 It supports open protocols, such as SOAP, and encapsulates the underlying
13 operating system and object model services. The framework provides a robust and
14 secure execution environment for the multiple programming languages and offers
15 secure, integrated class libraries.

16 The framework 132 is a multi-tiered architecture that includes an
17 application program interface (API) layer 142, a common language runtime (CLR)
18 layer 144, and an operating system/services layer 146. This layered architecture
19 allows updates and modifications to various layers without impacting other
20 portions of the framework. A common language specification (CLS) 140 allows
21 designers of various languages to write code that is able to access underlying
22 library functionality. The specification 140 functions as a contract between
23 language designers and library designers that can be used to promote language
24 interoperability. By adhering to the CLS, libraries written in one language can be
25 directly accessible to code modules written in other languages to achieve seamless

1 integration between code modules written in one language and code modules
2 written in another language. One exemplary detailed implementation of a CLS is
3 described in an ECMA standard created by participants in ECMA TC39/TG3.
4 The reader is directed to the ECMA web site at www.ecma.ch.

5 The API layer 142 presents groups of functions that the applications 130
6 can call to access the resources and services provided by layer 146. By exposing
7 the API functions for a network platform, application developers can create Web
8 applications for distributed computing systems that make full use of the network
9 resources and other Web services, without needing to understand the complex
10 interworkings of how those network resources actually operate or are made
11 available. Moreover, the Web applications can be written in any number of
12 programming languages, and translated into an intermediate language supported
13 by the common language runtime 144 and included as part of the common
14 language specification 140. In this way, the API layer 142 can provide methods
15 for a wide and diverse variety of applications.

16 Additionally, the framework 132 can be configured to support API calls
17 placed by remote applications executing remotely from the servers 134 that host
18 the framework. Representative applications 148(1) and 148(2) residing on clients
19 120(3) and 120(M), respectively, can use the API functions by making calls
20 directly, or indirectly, to the API layer 142 over the network 104.

21 The framework may also be implemented at the clients. Client 120(3)
22 represents the situation where a framework 150 is implemented at the client. This
23 framework may be identical to server-based framework 132, or modified for client
24 purposes. Alternatively, the client-based framework may be condensed in the
25 event that the client is a limited or dedicated function device, such as a cellular

1 phone, personal digital assistant, handheld computer, or other
2 communication/computing device.

4 DEVELOPERS' PROGRAMMING FRAMEWORK

5 Fig. 2 shows the programming framework 132 in more detail. The
6 common language specification (CLS) layer 140 supports applications written in a
7 variety of languages 130(1), 130(2), 130(3), 130(4), ..., 130(K). Such application
8 languages include Visual Basic, C++, C#, COBOL, Jscript, Perl, Eiffel, Python,
9 and so on. The common language specification 140 specifies a subset of features
10 or rules about features that, if followed, allow the various languages to
11 communicate. For example, some languages do not support a given type (e.g., an
12 "int*" type) that might otherwise be supported by the common language runtime
13 144. In this case, the common language specification 140 does not include the
14 type. On the other hand, types that are supported by all or most languages (e.g.,
15 the "int[]" type) is included in common language specification 140 so library
16 developers are free to use it and are assured that the languages can handle it. This
17 ability to communicate results in seamless integration between code modules
18 written in one language and code modules written in another language. Since
19 different languages are particularly well suited to particular tasks, the seamless
20 integration between languages allows a developer to select a particular language
21 for a particular code module with the ability to use that code module with modules
22 written in different languages. The common language runtime 144 allow seamless
23 multi-language development, with cross language inheritance, and provide a
24 robust and secure execution environment for the multiple programming languages.
25 For more information on the common language specification 140 and the common

1 language runtime 144, the reader is directed to co-pending applications entitled
2 “Method and System for Compiling Multiple Languages”, filed 6/21/2000 (serial
3 number 09/598,105) and “Unified Data Type System and Method” filed 7/10/2000
4 (serial number 09/613,289), which are incorporated by reference.

5 The framework 132 encapsulates the operating system 146(1) (e.g.,
6 Windows®-brand operating systems) and object model services 146(2) (e.g.,
7 Component Object Model (COM) or Distributed COM). The operating system
8 146(1) provides conventional functions, such as file management, notification,
9 event handling, user interfaces (e.g., windowing, menus, dialogs, etc.), security,
10 authentication, verification, processes and threads, memory management, and so
11 on. The object model services 146(2) provide interfacing with other objects to
12 perform various tasks. Calls made to the API layer 142 are handed to the common
13 language runtime layer 144 for local execution by the operating system 146(1)
14 and/or object model services 146(2).

15 The API 142 groups API functions into multiple namespaces. Namespaces
16 essentially define a collection of classes, interfaces, delegates, enumerations, and
17 structures, which are collectively called “types”, that provide a specific set of
18 related functionality. A class represents managed heap allocated data that has
19 reference assignment semantics. A delegate is an object oriented function pointer.
20 An enumeration is a special kind of value type that represents named constants. A
21 structure represents static allocated data that has value assignment semantics. An
22 interface defines a contract that other types can implement.

23 By using namespaces, a designer can organize a set of types into a
24 hierarchical namespace. The designer is able to create multiple groups from the
25 set of types, with each group containing at least one type that exposes logically

1 related functionality. In the exemplary implementation, the API 142 is organized
2 to include three root namespaces. It should be noted that although only three root
3 namespaces are illustrated in Fig. 2, additional root namespaces may also be
4 included in API 142. The three root namespaces illustrated in API 142 are: a first
5 namespace 200 for a presentation subsystem (which includes a namespace 202 for
6 a user interface shell), a second namespace 204 for web services, and a third
7 namespace 206 for a file system. Each group can then be assigned a name. For
8 instance, types in the presentation subsystem namespace 200 can be assigned the
9 name "Windows", and types in the file system namespace 206 can be assigned
10 names "Storage". The named groups can be organized under a single "global
11 root" namespace for system level APIs, such as an overall System namespace. By
12 selecting and prefixing a top level identifier, the types in each group can be easily
13 referenced by a hierarchical name that includes the selected top level identifier
14 prefixed to the name of the group containing the type. For instance, types in the
15 file system namespace 206 can be referenced using the hierarchical name
16 "System.Storage". In this way, the individual namespaces 200, 204, and 206
17 become major branches off of the System namespace and can carry a designation
18 where the individual namespaces are prefixed with a designator, such as a
19 "System." prefix.

20 The presentation subsystem namespace 200 pertains to programming and
21 content development. It supplies types that allow for the generation of
22 applications, documents, media presentations and other content. For example,
23 presentation subsystem namespace 200 provides a programming model that allows
24 developers to obtain services from the operating system 146(1) and/or object
25 model services 146(2).

1 The shell namespace 202 pertains to user interface functionality. It supplies
2 types that allow developers to embed user interface functionality in their
3 applications, and further allows developers to extend the user interface
4 functionality.

5 The web services namespace 204 pertains to an infrastructure for enabling
6 creation of a wide variety of applications, e.g. applications as simple as a chat
7 application that operates between two peers on an intranet, and/or as complex as a
8 scalable Web service for millions of users. The described infrastructure is
9 advantageously highly variable in that one need only use those parts that are
10 appropriate to the complexity of a particular solution. The infrastructure provides
11 a foundation for building message-based applications of various scale and
12 complexity. The infrastructure or framework provides APIs for basic messaging,
13 secure messaging, reliable messaging and transacted messaging. In the
14 embodiment described below, the associated APIs have been factored into a
15 hierarchy of namespaces in a manner that has been carefully crafted to balance
16 utility, usability, extensibility and versionability.

17 The file system namespace 206 pertains to storage. It supplies types that
18 allow for information storage and retrieval.

19 In addition to the framework 132, programming tools 210 are provided to
20 assist the developer in building Web services and/or applications. One example of
21 the programming tools 210 is Visual Studio™, a multi-language suite of
22 programming tools offered by Microsoft Corporation.

23 24 ROOT API NAMESPACES

Fig. 3 shows a portion of the presentation subsystem 200 in more detail. In one embodiment, the namespaces are identified according to a hierarchical naming convention in which strings of names are concatenated with periods. For instance, the presentation subsystem namespace 200 is identified by the root name "System.Windows". Within the "System.Windows" namespace is another namespace for various controls, identified as "System.Windows.Controls", which further identifies another namespace for primitives (not shown) known as "System.Windows.Controls.Primitives". With this naming convention in mind, the following provides a general overview of selected namespaces of the API 142, although other naming conventions could be used with equal effect.

As shown in Fig. 3, the presentation subsystem 200 includes multiple namespaces. The namespaces shown in Fig. 3 represent a particular embodiment of the presentation subsystem 200. Other embodiments of the presentation subsystem 200 may include one or more additional namespaces or may omit one or more of the namespaces shown in Fig. 3.

The presentation subsystem 200 is the root namespace for much of the presentation functionality of the API 142. A controls namespace 310 includes controls used to build a display of information, such as a user interface, and classes that allow a user to interact with an application. Example controls include "Button" that creates a button on the display, "RadioButton" that generates a radio-style button on the display, "Menu" that creates a menu on the display, "ToolBar" that creates a toolbar on the display, "Image" that generates an image on the display and "TreeView" that creates a hierarchical view of information.

Certain controls are created by nesting and arranging multiple elements. The controls have a logical model that hides the elements used to create the

1 controls, thereby simplifying the programming model. The controls can be styled
2 and themed by a developer or a user (e.g., by customizing the appearance and
3 behavior of user interface buttons). Some controls have addressable components
4 that allow an individual to adjust the style of individual controls. Additionally, the
5 controls can be sub-classed and extended by application developers and
6 component developers. The controls are rendered using vector graphics such that
7 they can be resized to fit the requirements of a particular interface or other display.
8 The controls are capable of utilizing animation to enhance, for example, the
9 interactive feel of a user interface and to show actions and reactions.

10 The controls namespace 310 includes one or more panels, which are
11 controls that measure and arrange their children (e.g., nested elements). For
12 example, a “DockPanel” panel arranges children by docking each child to the top,
13 left, bottom or right side of the display, and fills-in the remaining space with other
14 data. A particular panel may dock menus and toolbars to the top of the display, a
15 status bar to the bottom of the display, a folder list to the left side of the display,
16 and fills the rest of the space with a list of messages.

17 As mentioned above, System.Windows.Controls.Primitives is a namespace
18 that includes multiple controls that are components typically used by developers of
19 the controls in the System.Windows.Controls namespace and by developers
20 creating their own controls. Examples of these components include “Thumb and
21 RepeatButton”. “ScrollBar”, another component, is created using four repeat
22 buttons (one for “line up”, one for “line down”, one for “page up”, and one for
23 “page down”) and a “Thumb” for dragging the current view to another location in
24 the document. In another example, “ScrollViewer” is a control created using two
25 “ScrollBars” and one “ScrollArea” to provide a scrollable area.

1 The following list contains example classes exposed by the
2 System.Windows.Controls namespace. These classes allow a user to interact with,
3 for example, an application through various input and output capabilities as well
4 as additional display capabilities.

- 5 • AccessKey – AccessKey is a FrameworkElement element that wraps a
6 character, indicating that it is to receive keyboard cue decorations denoting
7 the character as a keyboard mnemonic. By default, the keyboard cue
8 decoration is an underline.
- 9 • Audio – Audio Element.
- 10 • Border – Draws a border, background, or both around another element.
- 11 • Button – Represents the standard button component that inherently reacts to
12 the Click event.
- 13 • Canvas – Defines an area within which a user can explicitly position child
14 elements by coordinates relative to the Canvas area.
- 15 • CheckBox – Use a CheckBox to give the user an option, such as true/false.
16 CheckBox allows the user to choose from a list of options. CheckBox
17 controls let the user pick a combination of options.
- 18 • CheckStateChangedEventArgs – This CheckStateChangedEventArgs class
19 contains additional information about the CheckStateChangedEvent event.
- 20 • CheckedChangedEventArgs – This CheckedChangedEventArgs class
21 contains additional information about the CheckedChangedEvent event.
- 22 • CheckStateChangedEventArgs – This CheckStateChangedEventArgs class
23 contains additional information about the CheckStateChangedEvent event.
- 24 • CheckStateChangedEventArgs – This CheckStateChangedEventArgs class
25 contains additional information about the CheckStateChangedEvent event.

1 ClickEventArgs – Contains information about the Click event.

2 ColumnStyle – Represents a changeable ColumnStyle object.

- 3
- 4 • ColumnStyles – Changeable pattern IList object that is a collection of
- 5 Changeable elements.
- 6 • ComboBox – ComboBox control.
- 7
- 8 • ComboBoxItem – Control that implements a selectable item inside a
- 9 ComboBox.
- 10 • ContactPickerDialog – Allows a user to select one or more contacts.
- 11
- 12 • ContactPropertyRequest – Allows an application to request information
- 13 about a contact property through a ContactPickerDialog. This class cannot
- 14 be inherited.
- 15 • ContactPropertyRequest Collection – Represents a collection of
- 16 ContactPropertyRequest objects.
- 17 • ContactSelection – Information about a selected contact from Microsoft®
- 18 Windows® File System, code-named "WinFS" or Microsoft Active
- 19 Directory®.
- 20 • ContactSelectionCollection – Represents a collection of ContactSelection
- 21 objects.
- 22
- 23 • ContactTextBox – An edit control that supports picking contacts or
- 24 properties of contacts.
- 25

1 ContactTextBoxSelectionChangedEventArgs – Arguments for the
2 ContactTextBoxSelectionChanged event.

- 3 • ContactTextBoxTextChangedEventArgs – Arguments for the
4 ContactTextBoxTextChanged event.
- 5 • ContactTextBoxTextResolvedEventArgs – Arguments for the
6 TextResolvedToContact event.
- 7 • ContentChangedEventArgs – The event arguments for
8 ContentChangedEvent.
- 9 • ContentControl – The base class for all controls with a single piece of
10 content.
- 11 • ContentPresenter – ContentPresenter is used within the style of a content
12 control to denote the place in the control's visual tree (chrome template)
13 where the content is to be added.
- 14 • ContextMenu – Control that defines a menu of choices for users to invoke.
- 15 • ContextMenuEventArgs – The data sent on a ContextMenuEvent.
- 16 • Control – Represents the base class for all user-interactive elements. This
17 class provides a base set of properties for its subclasses.
- 18 • Decorator – Base class for elements that apply effects onto or around a
19 single child element, such as Border.
- 20 • DockPanel – Defines an area within which you can arrange child elements
21 either horizontally or vertically, relative to each other.
- 22 • DockPanel – Defines an area within which you can arrange child elements
23 either horizontally or vertically, relative to each other.
- 24 • DockPanel – Defines an area within which you can arrange child elements
25 either horizontally or vertically, relative to each other.

1 DragDeltaEventArgs – This DragDeltaEventArgs class contains additional
2 information about the DragDeltaEvent event.

- 3 • FixedPanel – FixedPanel is the root element used in fixed-format
4 documents to contain fixed pages for pagination. FixedPanel displays
5 paginated content one page at a time or as a scrollable stack of pages.
6
- 7 • FlowPanel – FlowPanel is used to break, wrap, and align content that
8 exceeds the length of a single line. FlowPanel provides line-breaking and
9 alignment properties that can be used when the flow of the container's
10 content, Text for example, is likely to exceed the length of a single line.
- 11 • Frame – An area that can load the contents of another markup tree.
- 12 • Generator – Generator is the object that generates a UI on behalf of an
13 ItemsControl, working under the supervision of a GeneratorFactory.
- 14
- 15 • GeneratorFactory – A GeneratorFactory is responsible for generating the UI
16 on behalf of an ItemsControl. It maintains the association between the
17 items in the control's ItemsCollection (flattened view) and the
18 corresponding UIElements. The control's item-container can ask the
19 factory for a Generator, which does the actual generation of UI.
- 20 • GridPanel – Defines a grid area consisting of columns and rows.
- 21
- 22 • HeaderItemsControl – The base class for all controls that contain multiple
23 items and have a header.

24 HorizontalScrollBar – The Horizontal ScrollBar class.
25

1 HorizontalSlider – The Horizontal Slider class.

2 HyperLink – The HyperLink class implements navigation control. The
3 default presenter is TextPresenter.

- 4 • Image – Provides an easy way to include an image in a document or an
5 application.
- 6 • IncludeContactEventArgs – Arguments passed to handlers of the
7 ContactPickerDialog.IncludeContact event.
- 8 • ItemCollection – Maintains a collection of discrete items within a control.
9 Provides methods and properties that enable changing the collection
10 contents and obtaining data about the contents.
- 11 • ItemsChangedEventArgs – The ItemsChanged event is raised by a
12 GeneratorFactory to inform layouts that the items collection has changed.
- 13 • ItemsControl – The base class for all controls that have multiple children.
- 14 • ItemsView – ItemsView provides a flattened view of an ItemCollection.
- 15 • KeyboardNavigation – KeyboardNavigation class provide methods for
16 logical (Tab) and directional (arrow) navigation between focusable
17 controls.
- 18 • ListBox – Control that implements a list of selectable items.
- 19 • ListItem – Control that implements a selectable item inside a ListBox.
- 20
- 21
- 22
- 23

24 Menu – Control that defines a menu of choices for users to invoke.

25

MenuItem – A child item of Menu. MenuItems can be selected to invoke commands. MenuItems can be separators. MenuItems can be headers for submenus. MenuItems can be checked or unchecked.

- PageViewer – Represents a document-viewing composite control that contains a pagination control, a toolbar, and a page bar control.
- PaginationCompleteEventArgs – The event arguments for the `PaginationCompleteEvent`.
- PaginationProgressEventArgs – The event arguments for the `PaginationProgressEvent`.
- Pane – Provides a way to define window properties in a markup language (e.g., "XAML") without launching a new window.
- Panel – Provides a base class for all Panel elements. In order to instantiate a Panel element, use the derived concrete class.
- RadioButton – `RadioButton` implements an option button with two states: true or false.
- RadioButtonList – This control serves as a grouping control for `RadioButtons` and is the piece that handles `RadioButton` mutual exclusivity. The `RadioButtonList` inherits from `Selector`. The `RadioButtonList` is essentially a `Single SelectionMode Selector` and the concept of `Selection` (from `Selector`) is keyed off of the `Checked` property of the `RadioButton` it is grouping.
- RowStyle – Changeable pattern Changeable elements.

1 RowStyles – Changeable pattern IList object that is a collection of
2 Changeable elements.

- 3 • ScrollChangeEventArgs – The ScrollChangeEventArgs describe a change
4 in scrolling state.
- 5 • ScrollViewer –
- 6 • SelectedItemsCollection – A container for the selected items in a Selector.
- 7 • SelectionChangedEventArgs – The inputs to a selection changed event
8 handler.
- 9 • SimpleText – SimpleText is a lightweight, multi-line, single-format text
10 element intended for use in user interface (UI) scenarios. SimpleText
11 exposes several of the same formatting properties as Text and can often be
12 used for a performance gain at the cost of some versatility.
- 13 • StyleSelector – StyleSelector allows the app writer to provide custom style
14 selection logic. For example, with a class Bug as the Content, use a
15 particular style for Pri1 bugs and a different style for Pri2 bugs. An
16 application writer can override the SelectStyle method in a derived selector
17 class and assign an instance of this class to the StyleSelector property on
18 ContentPresenter class.
- 19 • Text – Represents a Text control that enables rendering of multiple formats
20 of Text. Text is best used within an application UI; more advanced text
21 scenarios benefit from the additional feature set of TextPanel. In most
22
23
24
25

1 cases where relatively simple text support is required, Text is the preferred
2 element because of its lightweight nature and range of features.

- 3 • TextBox – Represents the control that provides an editable region that
4 accepts text input.
- 5 • TextChangedEventArgs – The TextChangedEventArgs class represents a
6 type of RoutedEventArgs that are relevant to events raised by
7 TextRange.SetText().
- 8 • TextPanel – Formats, sizes, and draws text. TextPanel supports multiple
9 lines of text and multiple text formats.
- 10 • ToolTip – A control to display information when the user hovers over a
11 control.
- 12 • ToolTipEventArgs – The data sent on a ToolTipEvent.
- 13 • TransformDecorator – TransformDecorator contains a child and applies a
14 specified transform to it. TransformDecorator implements logic to measure
15 and arrange the child in its local (pre-transform) coordinates such that after
16 the transform, the child fits tightly within the decorator's space and uses
17 maximal area. The child therefore needs to have no knowledge that a
18 transform has been applied to it.
- 19 • UIElementCollection – A UIElementCollection is a ordered collection of
20 UIElements.

21 ValueChangedEventArgs – This ValueChangedEventArgs class contains
22 additional information about the ValueChangedEvent event.
23
24
25

1 VerticalScrollBar – The Vertical ScrollBar class.

2 VerticalSlider – The Vertical Slider class.

- 3
- 4 • Video – Plays a streaming video or audio file in a specified rectangle within
- 5 the current user coordinate system.
- 6 • VisibleChangedEventArgs – The VisibleChangedEventArgs class contains
- 7 additional information about the VisibleChangedEvent event.
- 8

9

10 The System.Windows.Controls namespace also contains various

11 enumerations. The following list contains example enumerations associated with

12 the System.Windows.Controls namespace.

- 13 • CharacterCase – Specifies the case of characters in a TextBox control when
- 14 the text is typed.
- 15
- 16 • CheckState – Specifies the state of a control, such as a check box, that can
- 17 be checked, unchecked, or set to an indeterminate state.
- 18 • ClickMode – Specifies when the Click event should fire.
- 19
- 20 • ContactControlPropertyPosition – Controls the position and display of the
- 21 property of the contact.
- 22 • ContactPickerDialogLayout – Specifies how the ContactPickerDialog
- 23 should display selected properties.
- 24
- 25

1 ContactPropertyCategory – Specifies which value to treat as the default in
2 the case where a property has multiple values for the user could choose
3 from. For example, if "Work" is specified as the preferred category when
4 requesting a phone number property from the ContactPickerDialog, and the
5 user selects a contact with both a work and home phone number, the work
6 phone number appears as the default selection. The user can then use the
7 UI to choose the home phone number instead.

- 8 • ContactPropertyType – Specifies a property of a contact that the
9 ContactPickerDialog can ask the user for.
- 10 • ContactType – Specifies which contact types to display in the
11 ContactPickerDialog.
- 12 • Direction – This enumeration is used by the GeneratorFactory and
13 Generator to specify the direction in which the generator produces UI.
- 14 • Dock – Specifies the Dock position of a child element within a DockPanel.
- 15 • GeneratorStatus – This enumeration is used by the GeneratorFactory to
16 indicate its status.
- 17 • KeyNavigationMode – The type of TabNavigation property specify how
18 the container will move the focus when Tab navigation occurs.
- 19 • MenuItemBehavior – Defines the different behaviors that a MenuItem
20 could have.
- 21 • MenuItemType – Defines the different placement types of MenuItems.

22 MenuItemType – Defines the different placement types of MenuItems.

- Orientation – Slider orientation types.
- PageViewerFit – Selects how pages should be fit into the PageViewer's Client area.
- PageViewerMode – Selects the current PageViewer mode Reflected in the mode dropdown.
- ScrollerVisibility – ScrollerVisiblty defines the visiblity behavior of a scrollbar.
- SelectionMode – Specifies the selection behavior for the ListBox.

“Position” is an example structure associated with the System.Windows.Controls namespace. A user of the Generator describes positions using this structure. For example: To start generating forward from the beginning of the item list, specify position (-1, 0) and direction Forward. To start generating backward from the end of the list, specify position (-1, 0) and direction Backward. To generate the items after the element with index k, specify position (k, 0) and direction Forward.

The following list contains example delegates associated with the System.Windows.Controls namespace.

- CheckedChangedEventHandler – This delegate is used by handlers of the CheckedChangedEvent event.

1 CheckStateChangedEventHandler – This delegate is used by handlers of
2 the CheckStateChangedEvent event.

- 3 • ClickEventHandler – Represents the methods that handle the Click event.
- 4
- 5 • ContactTextBoxSelectionChangedEventHandler – A delegate handler for
6 the ContactTextBoxSelectionChanged event.
- 7 • ContactTextBoxTextChangedEventHandler – A delegate handler for the
8 ContactTextBoxTextChanged event.
- 9 • ContactTextBoxTextResolvedEventHandler – A delegate handler for the
10 TextResolvedToContact event.
- 11
- 12 • ContentChangedDelegate – Delegate for the ContentChangedEvent.
- 13
- 14 • ContextMenuEventHandler - The callback type for handling a
15 ContextMenuEvent.
- 16 • DragDeltaEventHandler – This delegate is used by handlers of the
17 DragDeltaEvent event.
- 18 • IncludeContactEventHandler – Handler for
19 ContactPickerDialog.IncludeContact event.
- 20 • ItemsChangedEventHandler – The delegate to use for handlers that receive
21 ItemsChangedEventArgs.
- 22
- 23 • OpenedEventHandler – Handler for ContactPickerDialog.Opened event.
- 24 PaginationCompleteDelegate – Delegate for the PaginationCompleteEvent.
- 25

1 `PaginationProgressDelegate` – Delegate for the `PaginationProgressEvent`.

2 `ScrollChangeEventHandler` – This delegate is used by handlers of the
3 `ScrollChangeEvent` event.

- 4 • `SelectionChangedEventHandler` – The delegate type for handling a
5 selection changed event.
- 6 • `TextChangedEventHandler` – The delegate to use for handlers that receive
7 `TextChangedEventArgs`.
- 8 • `ToolTipEventHandler` – The callback type for handling a `ToolTipEvent`.
- 9 • `ValueChangedEventHandler` – This delegate is used by handlers of the
10 `ValueChangedEvent` event.
- 11 • `VisibleChangedEventHandler` – This delegate is used by handlers of the
12 `VisibleChangedEvent` event.
- 13 • `VisibleChangedEventHandler` – This delegate is used by handlers of the
14 `VisibleChangedEvent` event.

15
16
17 Another namespace, `System.Windows.Controls.Atoms`, is a sub-namespace
18 of the `System.Windows.Controls` namespace. `System.Windows.Controls.Atoms`
19 includes associated controls, event arguments and event handlers. The following
20 list contains example classes associated with the `System.Windows.Controls.Atoms`
21 namespace.

- 22 • `PageBar` – Represents a scrollable pagination control.
 - 23 • `PageElement` – Renders a specific page of paginated content. The page to
24 be rendered is specified by the `PageSource` property.
- 25

1 PageHoveredEventArgs – PageHoveredEventArgs provides information
2 about where the mouse pointer is hovering.

- 3 • PageScrolledEventArgs – The PageScrolledEventArgs contains info
4 pertaining to the PageScrolled Event.
- 5 • PageSelectedEventArgs – The PageSelectedEvent is fired when a new
6 row/column range selection is made.
- 7 • PageSelector – PageSelector: Allows the user to select a range of
8 rows/columns of pages to be displayed.
- 9 • PageSource – Identifies the source of the content to be paginated. It also
10 provides properties and methods for formatting paginated content.
- 11
- 12
- 13

14 The following list contains example delegates associated with the
15 System.Windows.Controls.Atoms namespace.

- 16 • PageHoveredEventHandler – This delegate is used by handlers of the
17 PageHoveredEvent event.
- 18 • PageScrolledEventHandler – This delegate is used by handlers of the
19 PageScrolled event.
- 20 • PageSelectedEventHandler – This delegate is used by handlers of the
21 PageSelectedEvent event.
- 22
- 23
- 24
- 25

1 A `System.Windows.Controls.Primitives` namespace is another sub-
2 namespace of the `System.Windows.Controls` namespace. As mentioned above,
3 the `Primitives` sub-namespace includes controls that are intended to be used as
4 primitives by other more complex controls. The following list contains example
5 classes associated with the `System.Windows.Controls.Primitives` namespace.

- 6 • `ButtonBase` – When overridden in a derived class, defines the relevant
7 events and properties, and provides handlers for the relevant input events.
- 8 • `Popup` – A control that creates a fly-out window that contains content.
- 9 • `RangeBase` – Represents the base class for elements that have a specific
10 range. Examples of such elements are scroll bars and progress bars. This
11 class defines the relevant events and properties, and provides handlers for
12 the events.
- 13 • `RepeatButton` – `RepeatButton` control adds repeating semantics of when the
14 Click event occurs.
- 15 • `ScrollArea` – `ScrollArea` is the effective element for scrolling. It contains
16 content that it clips and provides properties to expose the content's offset
17 and extent. It also provides default input handling such that scrolling can
18 be driven programatically or via keyboard or mouse wheel.
- 19 • `ScrollBar` – The `ScrollBar` class.
- 20 • `Selector` – The base class for controls that select items from among their
21 children.
- 22 • `Slider` – The `Slider` class.

1 Thumb – The thumb control enables basic drag-movement functionality for
2 scrollbars and window resizing widgets.

3
4
5 “IEnsureVisible” is an example interface associated with the
6 System.Windows.Controls.Primitives namespace. IEnsureVisible is implemented
7 on a visual to scroll/move a child visual into view.

8 The following list contains example enumerations associated with the
9 System.Windows.Controls.Primitives namespace.

- 10 • ArrowButtonStates –
- 11
- 12 • CloseModeType – Describes how a popup should behave to various mouse
- 13 events.
- 14
- 15 • Part – The Part enumeration is used to indicate the semantic use of the
- 16 controls that make up the scroll bar.
- 17
- 18 • PartStates – ScrollBar Part States.
- 19
- 20 • PlacementType – Describes where a popup should be placed on screen.
- 21
- 22 • SizeBoxStates –
- 23

24 A documents namespace 312 is a collection of semantic and formatting
25 elements that are used to create richly formatted and semantically rich documents.
In one embodiment, an “element” is a class that is primarily used in conjunction
with a hierarchy of elements (referred to as a “tree”). These elements can be

1 interactive (e.g., receiving user input via keyboard, mouse or other input device),
2 can render images or objects, and can assist with the arrangement of other
3 elements. Example elements include a “Block” element that implements a generic
4 block, a “Body” element that represents content that includes the body of a table, a
5 “Cell” element that contains tabular data within a table, a “Header” element that
6 represents the content included in the header of a table, and a “PageBreak”
7 element that is used to break content across multiple pages.

8 The following list contains example classes exposed by the
9 System.Windows.Documents namespace.

- 10 • AdaptiveMetricsContext - AdaptiveMetricsContext provides the root
11 element for adaptive-flow-format documents. Once a child panel is
12 encapsulated in an AdaptiveMetricsContext element, the content of the
13 panel is processed by the Reading Metrics Engine (RME). The size of the
14 child panel is used to calculate the number and size of any columns as well
15 as optimum font sizes and line heights.
- 16 • Block - Implements a generic block element that does not induce default
17 rendering behavior.
- 18 • BlockElement - Implements a base class for all Block elements.
- 19 • Body - Represents the content that comprises the body of a Table element.
- 20 • Bold - Implements a Bold element derived from Inline.
- 21
- 22
- 23
- 24
- 25

1 BreakRecord - Stores information necessary to continue formatting
2 paginated content across page breaks. Inherit from this class to provide
3 pagination support. This is an abstract class.

- 4 • Cell – Cells contain tabular data within a Table. Cell elements are
5 contained within a Row.
- 6
- 7 • CellCollection - Ordered collection of table cells.
- 8
- 9 • Column – The Column element is used to apportion the contents of a
10 GridPanel or Table.
- 11
- 12 • ColumnCollection - A ColumnCollection is an ordered collection of
13 Columns.
- 14
- 15 • ColumnResult - Represents a column's view-related information.
- 16
- 17 • ContainerParagraphResult – Provides access to calculated layout
18 parameters for a Paragraph object which contains only other Paragraph
19 objects.
- 20
- 21 • ContentPosition - Represents the position of content within a paragraph.
22 Inherit from this class to describe the position of associated content. This is
23 an abstract class.
- 24
- 25 • Document - The purpose of the Document class is to decouple the content
of a document from the UI "chrome" that surrounds it. "Decoupling"
means that you can author a document without thinking about (and without
committing to) its UI. The Document class holds document content,
typically a TextPanel or a FixedPanel and its children. A visual tree (by

1 default, a PageViewer) is associated with this element through the WPP
2 control styling mechanism.

- 3 • DocumentPage - Represents layout information for a control associated
4 with a page of a document subject to pagination. Inherit from this class to
5 implement to describe the layout information for these controls. This is an
6 abstract class.
- 7 • DocumentPageParagraphResult - Provides access to calculated layout
8 parameters for objects affected by pagination.
- 9 • FindEngine – Base class for find algorithms.
- 10 • FindEngineFactory – Find algorithms factory.
- 11 • FixedPage - Provides access to a single page of content within a fixed-
12 format layout document.
- 13 • Footer – Represents the content that comprises the footer of a Table
14 element.
- 15 • Header - Represents the content that comprises the header of a Table
16 element.
- 17 • Heading – Implements a block-level element that renders text as a heading.
- 18 • HyphenationDictionary - HyphenationDictionary represents a dictionary for
19 the purpose of providing hyphenation support within applications. It can
20 contain both an inline dictionary and a reference to an external dictionary.
- 21
- 22
- 23
- 24
- 25

1 The inline dictionary has higher priority and will be applied before entries
2 in the external dictionary.

- 3 • Hyphenator – The Hyphenator object maintains reference to hyphenation
4 data within a HyphenationDictionary and also performs hyphenation.
- 5 • Inline - Implements a generic Inline element that does not induce any
6 default rendering behavior.
- 7 • InlineElement - Implements a generic inline element as base class for all
8 inline elements.
- 9 • Italic – Implements an Italic element derived from Inline.
- 10 • LineBreak - Represents a markup element that forces a line break.
- 11 • LineResult - Provides access to calculated information of a line of text.
- 12 • List - Implements a List element. Lists are block-level elements designed
13 to be formatted with markers such as bullets or numbering.
- 14 • ListElementItem - Implements a ListElementItem, which supports markers
15 such as bullets or numbering.
- 16 • Note - Implements a Note element, which is analagous to the note element
17 in HTML.
- 18 • PageBreak - Represents a markup element used to break content across
19 various pages.
- 20
- 21
- 22
- 23
- 24
- 25

1 PageDescriptor - Implements PageDescriptor, which stores information
2 necessary to create paginated layout.

- 3 • Paragraph - Implements a block-level element used to render text in a
4 paragraph. Rendering behavior is analagous to that of the paragraph
5 element in HTML.
- 6 • ParagraphResult - Provides access to calculated layout parameters for a
7 Paragraph object.
- 8 • Row – Defines a row within a GridPanel or Table element.
- 9 • RowCollection – RowCollection represents an ordered collection of Rows.
- 10 • RowGroup – Specifies property defaults for a group of rows in a Table or
11 GridPanel.
- 12 • Section - Implements a generic container element. Rendering behavior is
13 analagous to the div element in HTML.
- 14 • SmallCaps - Implements an inline SmallCaps element. SmallCaps are
15 typographic forms that render as small capital versions of letters for
16 emphasis, as in a title.
- 17 • Subscript - Represents an inline Subscript element. Subscript characters
18 are written immediately below, below and to the left, or below and to the
19 right of other characters.
- 20 • Subscript - Represents an inline Subscript element. Subscript characters
21 are written immediately below, below and to the left, or below and to the
22 right of other characters.
- 23 • Subscript - Represents an inline Subscript element. Subscript characters
24 are written immediately below, below and to the left, or below and to the
25 right of other characters.

1 Superscript - Represents an inline Superscript element. Superscript
2 characters are typically letters or numbers and render immediately above,
3 above and to the left, or above and to the right of other characters.

- 4 • Table – Table is used to display complex data in tabular form using a
5 markup language (e.g., "XAML").
- 6
- 7 • TextArray - Base API for text access and manipulation.
- 8
- 9 • TextChangedEventArgs - The TextChangedEventArgs defines the event
10 arguments sent when a TextArray is changed.
- 11
- 12 • TextElement - TextElement provides TextRange facilities for the TextTree.
13 It is an immutable, continuous TextRange with fixed endpoints. It provides
14 ContentElement Input, Focus and Eventing support. It also provides
15 DependencyObject property support.
- 16
- 17 • TextNavigator - This can enumerate text content. Implements a movable
18 TextPosition. It can move by text run or be positioned at a know location
19 in text.
- 20
- 21 • TextParagraphResult - Provides access to calculated layout parameters for
22 text, including floated objects and figures.
- 23
- 24 • TextPosition - This is an object representing a certain position in a
25 TextArray. A compact object representing a position in text automatically
maintains position when text changes. Comparison operations are only
applicable to positions within same TextArray (same Context) TextPosition
can be static or movable. IsChangeable property tells the kind of position.

1 TextRange - TextRange is an abstract class providing generic association of
2 zero or more subranges with properties. Subrange manipulation is defined
3 on derived classes.

- 4 • TextRangeMovable – TextRangeMovable is an abstract class for movable
5 TextRanges. It adds the ability to move the start and end points based on
6 TextUnits.
- 7 • TextTreeChangedEventArgs - The TextChangedEventArgs defines the
8 event arguments sent when a TextArray is changed.
- 9 • TextTreeDumper - TreeDumper is a tree test class that is public due to
10 packaging issues.
- 11 • TextTreeNavigator - This is an object representing a certain moveable
12 position in a TextTree. It is a specific implementation of TextNavigator for
13 use only in the TextTree.
- 14 • TextTreePosition - This is an object representing a certain immutable
15 position in a TextTree. It is a specific implementation of TextPosition for
16 use only in the TextTree.
- 17 • TextTreeRange - Provides TextRange facilities for the TextTree. It is a
18 mutable, continuous TextRange with movable endpoints.
- 19 • TextTreeRangeContentEnumerator - Enumerator on object children directly
20 under a TextTreeRange.
- 21 • TextTreeRangeContentEnumerator - Enumerator on object children directly
22 under a TextTreeRange.
- 23 • TextTreeRangeContentEnumerator - Enumerator on object children directly
24 under a TextTreeRange.

25 TextUnit - Extensible unit of text navigation.

1 TextUnits - Commonly used text units for TextPosition and TextRange.

2 Typography - Provides access to a rich set of OpenType typography
3 properties.

- 4
- 5 • UIElementParagraphResult - The ParagraphResult for a paragraph which is
6 composed entirely of a UIElement. Used for Floaters, Figures and
7 embedded block level UIElements.
- 8 • Underline – Implements an Underline element derived from InlineElement.
- 9

10

11 The following list contains example interfaces associated with the
12 System.Windows.Documents namespace.

- 13 • IDocumentContentHost - Implement this interface on a content host so that
14 children of that host can notify the host when content is changing.
- 15
- 16 • IDocumentFormatter - Implement this interface on an element to provide
17 support for document features such as pagination.
- 18 • ITextDocumentResult - Implement this interface to maintain column
19 information for a document.
- 20 • ITextParagraphResult - Implement this interface to provide text and
21 positioning information for text paragraphs.
- 22
- 23
- 24
- 25

1 The following list contains example enumerations associated with the
2 System.Windows.Documents namespace.

- 3 • ElementEdge - This identifies the edge of an object where a TextPosition is
4 located.
- 5 • FindAdvancedOptions - The advanced search options used by
6 FindAlgorithm (search initialization) and
7 TextRangeMovable/TextSelection (simplified search execution) classes.
8
- 9 • FindOptions - The simplified search options used by TextBox.Find
10 methods.
- 11 • LogicalDirection - LogicalDirection defines a logical direction for
12 movement in text. It is also used to determine where a TextPosition will
13 move when content is inserted at the TextPosition.
- 14 • TextArrayRunType - This identifies the run where a TextPosition is
15 located, taking LogicalDirection into account.
- 16 • TextChangeOptions - Possible text changes for CanChangeText.
- 17 • TextMoveOptions - This controls the movement of TextNavigator by
18 specifying conditions to halt navigation.
19
- 20
- 21
- 22

23 The following list contains example delegates associated with the
24 System.Windows.Documents namespace.
25

ObjectCloneDelegate - Callback method to provide a clone or copy of a DependencyObject when a portion of a TextArray is being copied or moved.

- TextChangedEventHandler - The TextChangedEventHandler delegate is called with TextChangedEventArgs every time content is added to or removed from the TextTree.

A shapes namespace 314 is a collection of vector graphics elements used to create images and objects. The use of vector graphics elements allows the elements to be easily resized to fit the requirements of a particular interface or display device. The following list contains example classes exposed by the System.Windows.Shapes namespace.

- Ellipse – Draws an ellipse.
- Glyphs - Represents a glyph shape in a markup language such as "XAML". Glyphs are used to represent fonts.
- Line – Draws a straight line between two points.
- Path – Draws a series of connected lines and curves.
- Polygon – Draws a polygon (a connected series of lines that forms a closed shape).
- Polyline – Draws a series of connected straight lines.
- Rectangle – Draws a rectangle.

1 Shape – An abstract class that provides base functionality for shape
2 elements, such as ellipse, polygon and rectangle.

3
4 The System.Windows.Controls, System.Windows.Documents and
5 System.Windows.Shapes namespaces provide an integrated system for developing
6 applications and related components. This integrated system provides a common
7 programming model for all three namespaces, thereby simplifying development of
8 application programs. This interoperability among all three namespaces allows
9 developers to learn a single programming architecture that is applied to any of the
10 features provided by three namespaces. For example, a common markup language
11 is used across all three namespaces. This common markup language provides for
12 the mapping of classes and properties specified in XML markup to an instantiated
13 tree of objects.

14 Additionally, a consistent programming model and consistent services are
15 used across the three namespaces. For example, a consistent event system is used
16 to initiate and process various events. A common property system is used to style
17 various properties, bind data to a property, or animate a property, regardless of
18 whether the property is associated with the “Controls”, “Documents”, or “Shapes”
19 namespace. Additionally, the same input paradigms and layout handling is
20 common across all three namespaces. For example, various controls from the
21 System.Windows.Controls namespace can be nested in the middle of a document’s
22 content defined using the System.Windows.Documents namespace.

23 Example source files will include a set of windows and panes (also referred
24 to as “pages”) that are declaratively defined using “Controls”, “Documents” and
25 “Shapes”. Interaction logic is also provided for the windows and panes. The

1 interaction logic identifies program code that is executed in response to a
2 particular user action or in response to occurrence of an event or activity. The
3 interaction logic is defined, for example, using a Common Language Runtime
4 (CLR) language. A CLR is a runtime environment that handles execution of
5 program code (e.g., .NET program code) and provides various services such as
6 security-related services and memory-related services. Example CLR languages
7 include C# and visual basic. Source files may also include other stand-alone
8 programming language files, such as C# or visual basic files.

9 Although this discussion refers to the integration of the “Controls”,
10 “Documents” and “Shapes” namespaces, this integration may be applied to any or
11 all of the namespaces and sub-namespaces discussed herein.

12 A data namespace 316 includes classes and interfaces used to bind
13 properties of elements to data sources, data source classes, and data-specific
14 implementations of data collections and views. These classes and interfaces are
15 also used to handle exceptions in data entry and allow runtime creation of a user
16 interface based on information in various data sources. Data can be displayed in
17 textual form or can be utilized to change the formatting of the display, such as
18 displaying dollar amounts in red if they are negative. Example classes include a
19 “Bind” class that represents a binding declaration object that manages bindings
20 between a dynamic property user interface and source data, and an
21 “XmlDataSource” class that serves as a data source for data binding to XML
22 content nodes.

23 A media namespace 318 provides various media classes. Application
24 developers as well as component developers may use these classes to develop
25 various presentation functionality. Example classes in media namespace 318

1 include an “ImageEffect” class that permits certain imaging effects (e.g., blur and
2 grayscale), and a “Brush” class that provides a mechanism for filling an area using
3 solid colors, gradients, images, video, and the like.

4 The media namespace 318 includes a sub-namespace
5 System.Windows.Media.Animation that includes services that allow a developer
6 to animate properties and coordinate a set of animations with a set of timelines.
7 An animation is an object that changes a value over a period of time. Animation
8 effects include moving an object on the display, and changing the size, shape, or
9 color of an object. Multiple animation classes are provided to implement various
10 animation effects. Effects can be achieved by associating an animation with an
11 element’s property value. For example, to create a rectangle that fades in and out
12 of view, one or more animations are associated with the opacity property of the
13 rectangle.

14 The media namespace 318 also includes a sub-namespace
15 System.Windows.Media.TextFormatting that provides various text services. For
16 example, a “TextFormatter” text engine provides services for breaking text lines
17 and formatting text presented on a display. “TextFormatter” is capable of handling
18 different text character formats and paragraph styles as well as handling
19 international text layout.

20 A design namespace 320 provides classes that enable the editing of forms
21 and text, formatting data and cross-process data sharing. These classes provide an
22 extensible framework for editing documents, applications, and other content.

23 An input namespace 322 includes an input manager that coordinates inputs
24 received by the system. The input namespace 322 also includes classes that help
25

1 manage and provide control for different input devices, such as a keyboard or a
2 mouse.

3 A navigation namespace 324 provides a set of classes and services that
4 allow the building of applications with navigation paradigms, such as a browser
5 application. These classes and services permit the development of applications
6 with customized navigation experiences. For example, when purchasing a product
7 or service from an online merchant, clicking a “Back” button causes the
8 application to display a different page that asks the user if they want to cancel or
9 change their order. In another example, activating a “Refresh” button causes an
10 application to retrieve new data instead of first reloading the application followed
11 by retrieving the new data. The navigation namespace 324 also includes page
12 functions that provide a mechanism for generating a hierarchy of questions that are
13 presented to a user.

14 An automation namespace 326 provides a set of classes that support
15 accessibility and user interface automation.

16 A serialization namespace 328 provides a parser that can load or save a
17 hierarchy of objects (e.g., elements) from or to an XML file or a file with a binary
18 representation. This process also sets properties associated with the objects and
19 associates event handlers.

20 An interop namespace 330 provides a set of classes that enable
21 interoperability with other operating systems or computing platforms.

22 A forms.interop namespace 332 provides an element that allows an
23 application to host a form control operation.

24 Another namespace, System.IO.CompoundFile (not shown in Fig. 3)
25 provides services to utilize a compound file in which various document

1 distributable files are stored. These services allow for the encryption and
2 compression of content. The services also support the storage of multiple
3 renditions of the same content, such as a re-flowable document and a fixed-format
4 document.

6 EXEMPLARY COMPUTING SYSTEM AND ENVIRONMENT

7 Fig. 4 illustrates an example of a suitable computing environment 400
8 within which the programming framework 132 may be implemented (either fully
9 or partially). The computing environment 400 may be utilized in the computer
10 and network architectures described herein.

11 The exemplary computing environment 400 is only one example of a
12 computing environment and is not intended to suggest any limitation as to the
13 scope of use or functionality of the computer and network architectures. Neither
14 should the computing environment 400 be interpreted as having any dependency
15 or requirement relating to any one or combination of components illustrated in the
16 exemplary computing environment 400.

17 The framework 132 may be implemented with numerous other general
18 purpose or special purpose computing system environments or configurations.
19 Examples of well known computing systems, environments, and/or configurations
20 that may be suitable for use include, but are not limited to, personal computers,
21 server computers, multiprocessor systems, microprocessor-based systems, network
22 PCs, minicomputers, mainframe computers, distributed computing environments
23 that include any of the above systems or devices, and so on. Compact or subset
24 versions of the framework may also be implemented in clients of limited
25

1 resources, such as cellular phones, personal digital assistants, handheld computers,
2 or other communication/computing devices.

3 The framework 132 may be described in the general context of computer-
4 executable instructions, such as program modules, being executed by one or more
5 computers or other devices. Generally, program modules include routines,
6 programs, objects, components, data structures, etc. that perform particular tasks
7 or implement particular abstract data types. The framework 132 may also be
8 practiced in distributed computing environments where tasks are performed by
9 remote processing devices that are linked through a communications network. In
10 a distributed computing environment, program modules may be located in both
11 local and remote computer storage media including memory storage devices.

12 The computing environment 400 includes a general-purpose computing
13 device in the form of a computer 402. The components of computer 402 can
14 include, by are not limited to, one or more processors or processing units 404, a
15 system memory 406, and a system bus 408 that couples various system
16 components including the processor 404 to the system memory 406.

17 The system bus 408 represents one or more of several possible types of bus
18 structures, including a memory bus or memory controller, a peripheral bus, an
19 accelerated graphics port, and a processor or local bus using any of a variety of
20 bus architectures. By way of example, such architectures can include an Industry
21 Standard Architecture (ISA) bus, a Micro Channel Architecture (MCA) bus, an
22 Enhanced ISA (EISA) bus, a Video Electronics Standards Association (VESA)
23 local bus, and a Peripheral Component Interconnects (PCI) bus also known as a
24 Mezzanine bus.

1 Computer 402 typically includes a variety of computer readable media.
2 Such media can be any available media that is accessible by computer 402 and
3 includes both volatile and non-volatile media, removable and non-removable
4 media.

5 The system memory 406 includes computer readable media in the form of
6 volatile memory, such as random access memory (RAM) 410, and/or non-volatile
7 memory, such as read only memory (ROM) 412. A basic input/output system
8 (BIOS) 414, containing the basic routines that help to transfer information
9 between elements within computer 402, such as during start-up, is stored in ROM
10 412. RAM 410 typically contains data and/or program modules that are
11 immediately accessible to and/or presently operated on by the processing unit 404.

12 Computer 402 may also include other removable/non-removable,
13 volatile/non-volatile computer storage media. By way of example, Fig. 4
14 illustrates a hard disk drive 416 for reading from and writing to a non-removable,
15 non-volatile magnetic media (not shown), a magnetic disk drive 418 for reading
16 from and writing to a removable, non-volatile magnetic disk 420 (e.g., a “floppy
17 disk”), and an optical disk drive 422 for reading from and/or writing to a
18 removable, non-volatile optical disk 424 such as a CD-ROM, DVD-ROM, or other
19 optical media. The hard disk drive 416, magnetic disk drive 418, and optical disk
20 drive 422 are each connected to the system bus 408 by one or more data media
21 interfaces 426. Alternatively, the hard disk drive 416, magnetic disk drive 418,
22 and optical disk drive 422 can be connected to the system bus 408 by one or more
23 interfaces (not shown).

24 The disk drives and their associated computer-readable media provide non-
25 volatile storage of computer readable instructions, data structures, program

1 modules, and other data for computer 402. Although the example illustrates a hard
2 disk 416, a removable magnetic disk 420, and a removable optical disk 424, it is to
3 be appreciated that other types of computer readable media which can store data
4 that is accessible by a computer, such as magnetic cassettes or other magnetic
5 storage devices, flash memory cards, CD-ROM, digital versatile disks (DVD) or
6 other optical storage, random access memories (RAM), read only memories
7 (ROM), electrically erasable programmable read-only memory (EEPROM), and
8 the like, can also be utilized to implement the exemplary computing system and
9 environment.

10 Any number of program modules can be stored on the hard disk 416,
11 magnetic disk 420, optical disk 424, ROM 412, and/or RAM 410, including by
12 way of example, an operating system 426, one or more application programs 428,
13 other program modules 430, and program data 432. Each of the operating system
14 426, one or more application programs 428, other program modules 430, and
15 program data 432 (or some combination thereof) may include elements of the
16 programming framework 132.

17 A user can enter commands and information into computer 402 via input
18 devices such as a keyboard 434 and a pointing device 436 (e.g., a "mouse").
19 Other input devices 438 (not shown specifically) may include a microphone,
20 joystick, game pad, satellite dish, serial port, scanner, and/or the like. These and
21 other input devices are connected to the processing unit 404 via input/output
22 interfaces 440 that are coupled to the system bus 408, but may be connected by
23 other interface and bus structures, such as a parallel port, game port, or a universal
24 serial bus (USB).
25

1 A monitor 442 or other type of display device can also be connected to the
2 system bus 408 via an interface, such as a video adapter 444. In addition to the
3 monitor 442, other output peripheral devices can include components such as
4 speakers (not shown) and a printer 446 which can be connected to computer 402
5 via the input/output interfaces 440.

6 Computer 402 can operate in a networked environment using logical
7 connections to one or more remote computers, such as a remote computing device
8 448. By way of example, the remote computing device 448 can be a personal
9 computer, portable computer, a server, a router, a network computer, a peer device
10 or other common network node, and so on. The remote computing device 448 is
11 illustrated as a portable computer that can include many or all of the elements and
12 features described herein relative to computer 402.

13 Logical connections between computer 402 and the remote computer 448
14 are depicted as a local area network (LAN) 450 and a general wide area network
15 (WAN) 452. Such networking environments are commonplace in offices,
16 enterprise-wide computer networks, intranets, and the Internet.

17 When implemented in a LAN networking environment, the computer 402 is
18 connected to a local network 450 via a network interface or adapter 454. When
19 implemented in a WAN networking environment, the computer 402 typically
20 includes a modem 456 or other means for establishing communications over the
21 wide network 452. The modem 456, which can be internal or external to computer
22 402, can be connected to the system bus 408 via the input/output interfaces 440 or
23 other appropriate mechanisms. It is to be appreciated that the illustrated network
24 connections are exemplary and that other means of establishing communication
25 link(s) between the computers 402 and 448 can be employed.

1 In a networked environment, such as that illustrated with computing
2 environment 400, program modules depicted relative to the computer 402, or
3 portions thereof, may be stored in a remote memory storage device. By way of
4 example, remote application programs 458 reside on a memory device of remote
5 computer 448. For purposes of illustration, application programs and other
6 executable program components such as the operating system are illustrated herein
7 as discrete blocks, although it is recognized that such programs and components
8 reside at various times in different storage components of the computing device
9 402, and are executed by the data processor(s) of the computer.

10 An implementation of the framework 132, and particularly, the API 142 or
11 calls made to the API 142, may be stored on or transmitted across some form of
12 computer readable media. Computer readable media can be any available media
13 that can be accessed by a computer. By way of example, and not limitation,
14 computer readable media may comprise "computer storage media" and
15 "communications media." "Computer storage media" include volatile and non-
16 volatile, removable and non-removable media implemented in any method or
17 technology for storage of information such as computer readable instructions, data
18 structures, program modules, or other data. Computer storage media includes, but
19 is not limited to, RAM, ROM, EEPROM, flash memory or other memory
20 technology, CD-ROM, digital versatile disks (DVD) or other optical storage,
21 magnetic cassettes, magnetic tape, magnetic disk storage or other magnetic storage
22 devices, or any other medium which can be used to store the desired information
23 and which can be accessed by a computer.

24 "Communication media" typically embodies computer readable
25 instructions, data structures, program modules, or other data in a modulated data

1 signal, such as carrier wave or other transport mechanism. Communication media
2 also includes any information delivery media. The term “modulated data signal”
3 means a signal that has one or more of its characteristics set or changed in such a
4 manner as to encode information in the signal. By way of example, and not
5 limitation, communication media includes wired media such as a wired network or
6 direct-wired connection, and wireless media such as acoustic, RF, infrared, and
7 other wireless media. Combinations of any of the above are also included within
8 the scope of computer readable media.

9 Alternatively, portions of the framework may be implemented in hardware
10 or a combination of hardware, software, and/or firmware. For example, one or
11 more application specific integrated circuits (ASICs) or programmable logic
12 devices (PLDs) could be designed or programmed to implement one or more
13 portions of the framework.

14 A programming interface (or more simply, interface) may be viewed as any
15 mechanism, process, protocol for enabling one or more segment(s) of code to
16 communicate with or access the functionality provided by one or more other
17 segment(s) of code. Alternatively, a programming interface may be viewed as one
18 or more mechanism(s), method(s), function call(s), module(s), object(s), etc. of a
19 component of a system capable of communicative coupling to one or more
20 mechanism(s), method(s), function call(s), module(s), etc. of other component(s).
21 The term “segment of code” in the preceding sentence is intended to include one
22 or more instructions or lines of code, and includes, e.g., code modules, objects,
23 subroutines, functions, and so on, regardless of the terminology applied or whether
24 the code segments are separately compiled, or whether the code segments are
25 provided as source, intermediate, or object code, whether the code segments are

1 utilized in a runtime system or process, or whether they are located on the same or
2 different machines or distributed across multiple machines, or whether the
3 functionality represented by the segments of code are implemented wholly in
4 software, wholly in hardware, or a combination of hardware and software.

5 Notionally, a programming interface may be viewed generically, as shown
6 in Fig. 5 or Fig. 6. Fig. 5 illustrates an interface Interface1 as a conduit through
7 which first and second code segments communicate. Fig. 6 illustrates an interface
8 as comprising interface objects I1 and I2 (which may or may not be part of the
9 first and second code segments), which enable first and second code segments of a
10 system to communicate via medium M. In the view of Fig. 6, one may consider
11 interface objects I1 and I2 as separate interfaces of the same system and one may
12 also consider that objects I1 and I2 plus medium M comprise the interface.
13 Although Figs. 5 and 6 show bi-directional flow and interfaces on each side of the
14 flow, certain implementations may only have information flow in one direction (or
15 no information flow as described below) or may only have an interface object on
16 one side. By way of example, and not limitation, terms such as application
17 programming or program interface (API), entry point, method, function,
18 subroutine, remote procedure call, and component object model (COM) interface,
19 are encompassed within the definition of programming interface.

20 Aspects of such a programming interface may include the method whereby
21 the first code segment transmits information (where "information" is used in its
22 broadest sense and includes data, commands, requests, etc.) to the second code
23 segment; the method whereby the second code segment receives the information;
24 and the structure, sequence, syntax, organization, schema, timing and content of
25 the information. In this regard, the underlying transport medium itself may be

1 unimportant to the operation of the interface, whether the medium be wired or
2 wireless, or a combination of both, as long as the information is transported in the
3 manner defined by the interface. In certain situations, information may not be
4 passed in one or both directions in the conventional sense, as the information
5 transfer may be either via another mechanism (e.g. information placed in a buffer,
6 file, etc. separate from information flow between the code segments) or non-
7 existent, as when one code segment simply accesses functionality performed by a
8 second code segment. Any or all of these aspects may be important in a given
9 situation, e.g., depending on whether the code segments are part of a system in a
10 loosely coupled or tightly coupled configuration, and so this list should be
11 considered illustrative and non-limiting.

12 This notion of a programming interface is known to those skilled in the art
13 and is clear from the foregoing detailed description of the invention. There are,
14 however, other ways to implement a programming interface, and, unless expressly
15 excluded, these too are intended to be encompassed by the claims set forth at the
16 end of this specification. Such other ways may appear to be more sophisticated or
17 complex than the simplistic view of Figs. 5 and 6, but they nonetheless perform a
18 similar function to accomplish the same overall result. We will now briefly
19 describe some illustrative alternative implementations of a programming interface.

20 21 A. FACTORING

22 A communication from one code segment to another may be accomplished
23 indirectly by breaking the communication into multiple discrete communications.
24 This is depicted schematically in Figs. 7 and 8. As shown, some interfaces can be
25 described in terms of divisible sets of functionality. Thus, the interface

1 functionality of Figs. 5 and 6 may be factored to achieve the same result, just as
2 one may mathematically provide 24, or 2 times 2 times 3 times 2. Accordingly, as
3 illustrated in Fig. 7, the function provided by interface Interface1 may be
4 subdivided to convert the communications of the interface into multiple interfaces
5 Interface1A, Interface 1B, Interface 1C, etc. while achieving the same result. As
6 illustrated in Fig. 8, the function provided by interface I1 may be subdivided into
7 multiple interfaces I1a, I1b, I1c, etc. while achieving the same result. Similarly,
8 interface I2 of the second code segment which receives information from the first
9 code segment may be factored into multiple interfaces I2a, I2b, I2c, etc. When
10 factoring, the number of interfaces included with the 1st code segment need not
11 match the number of interfaces included with the 2nd code segment. In either of the
12 cases of Figs. 7 and 8, the functional spirit of interfaces Interface1 and I1 remain
13 the same as with Figs. 5 and 6, respectively. The factoring of interfaces may also
14 follow associative, commutative, and other mathematical properties such that the
15 factoring may be difficult to recognize. For instance, ordering of operations may
16 be unimportant, and consequently, a function carried out by an interface may be
17 carried out well in advance of reaching the interface, by another piece of code or
18 interface, or performed by a separate component of the system. Moreover, one of
19 ordinary skill in the programming arts can appreciate that there are a variety of
20 ways of making different function calls that achieve the same result.

21 22 B. REDEFINITION

23 In some cases, it may be possible to ignore, add or redefine certain aspects
24 (e.g., parameters) of a programming interface while still accomplishing the
25 intended result. This is illustrated in Figs. 9 and 10. For example, assume interface

Interface1 of Fig. 5 includes a function call *Square(input, precision, output)*, a call that includes three parameters, *input*, *precision* and *output*, and which is issued from the 1st Code Segment to the 2nd Code Segment., If the middle parameter *precision* is of no concern in a given scenario, as shown in Fig. 9, it could just as well be ignored or even replaced with a *meaningless* (in this situation) parameter. One may also add an *additional* parameter of no concern. In either event, the functionality of square can be achieved, so long as output is returned after input is squared by the second code segment. *Precision* may very well be a meaningful parameter to some downstream or other portion of the computing system; however, once it is recognized that *precision* is not necessary for the narrow purpose of calculating the square, it may be replaced or ignored. For example, instead of passing a valid *precision* value, a meaningless value such as a birth date could be passed without adversely affecting the result. Similarly, as shown in Fig. 10, interface I1 is replaced by interface I1', redefined to ignore or add parameters to the interface. Interface I2 may similarly be redefined as interface I2', redefined to ignore unnecessary parameters, or parameters that may be processed elsewhere. The point here is that in some cases a programming interface may include aspects, such as parameters, that are not needed for some purpose, and so they may be ignored or redefined, or processed elsewhere for other purposes.

C. INLINE CODING

It may also be feasible to merge some or all of the functionality of two separate code modules such that the "interface" between them changes form. For example, the functionality of Figs. 5 and 6 may be converted to the functionality of Figs. 11 and 12, respectively. In Fig. 11, the previous 1st and 2nd Code Segments

1 of Fig. 5 are merged into a module containing both of them. In this case, the code
2 segments may still be communicating with each other but the interface may be
3 adapted to a form which is more suitable to the single module. Thus, for example,
4 formal Call and Return statements may no longer be necessary, but similar
5 processing or response(s) pursuant to interface Interface1 may still be in effect.
6 Similarly, shown in Fig. 12, part (or all) of interface I2 from Fig. 6 may be written
7 inline into interface I1 to form interface I1". As illustrated, interface I2 is divided
8 into I2a and I2b, and interface portion I2a has been coded in-line with interface I1
9 to form interface I1". For a concrete example, consider that the interface I1 from
10 Fig. 6 performs a function call square (*input*, *output*), which is received by
11 interface I2, which after processing the value passed with *input* (to square it) by
12 the second code segment, passes back the squared result with *output*. In such a
13 case, the processing performed by the second code segment (squaring *input*) can
14 be performed by the first code segment without a call to the interface.

15 16 D. DIVORCE

17 A communication from one code segment to another may be accomplished
18 indirectly by breaking the communication into multiple discrete communications.
19 This is depicted schematically in Figs. 13 and 14. As shown in Fig. 13, one or
20 more piece(s) of middleware (Divorce Interface(s), since they divorce
21 functionality and / or interface functions from the original interface) are provided
22 to convert the communications on the first interface, Interface1, to conform them
23 to a different interface, in this case interfaces Interface2A, Interface2B and
24 Interface2C. This might be done, e.g., where there is an installed base of
25 applications designed to communicate with, say, an operating system in

1 accordance with an Interface1 protocol, but then the operating system is changed
2 to use a different interface, in this case interfaces Interface2A, Interface2B and
3 Interface2C. The point is that the original interface used by the 2nd Code Segment
4 is changed such that it is no longer compatible with the interface used by the 1st
5 Code Segment, and so an intermediary is used to make the old and new interfaces
6 compatible. Similarly, as shown in Fig. 14, a third code segment can be introduced
7 with divorce interface DI1 to receive the communications from interface I1 and
8 with divorce interface DI2 to transmit the interface functionality to, for example,
9 interfaces I2a and I2b, redesigned to work with DI2, but to provide the same
10 functional result. Similarly, DI1 and DI2 may work together to translate the
11 functionality of interfaces I1 and I2 of Fig. 6 to a new operating system, while
12 providing the same or similar functional result.

13 14 E. REWRITING

15 Yet another possible variant is to dynamically rewrite the code to replace
16 the interface functionality with something else but which achieves the same
17 overall result. For example, there may be a system in which a code segment
18 presented in an intermediate language (e.g. Microsoft IL, Java ByteCode, etc.) is
19 provided to a Just-in-Time (JIT) compiler or interpreter in an execution
20 environment (such as that provided by the .Net framework, the Java runtime
21 environment, or other similar runtime type environments). The JIT compiler may
22 be written so as to dynamically convert the communications from the 1st Code
23 Segment to the 2nd Code Segment, i.e., to conform them to a different interface as
24 may be required by the 2nd Code Segment (either the original or a different 2nd
25 Code Segment). This is depicted in Figs. 15 and 16. As can be seen in Fig. 15, this

1 approach is similar to the Divorce scenario described above. It might be done, e.g.,
2 where an installed base of applications are designed to communicate with an
3 operating system in accordance with an Interface 1 protocol, but then the operating
4 system is changed to use a different interface. The JIT Compiler could be used to
5 conform the communications on the fly from the installed-base applications to the
6 new interface of the operating system. As depicted in Fig. 16, this approach of
7 dynamically rewriting the interface(s) may be applied to dynamically factor, or
8 otherwise alter the interface(s) as well.

9 It is also noted that the above-described scenarios for achieving the same or
10 similar result as an interface via alternative embodiments may also be combined in
11 various ways, serially and/or in parallel, or with other intervening code. Thus, the
12 alternative embodiments presented above are not mutually exclusive and may be
13 mixed, matched and combined to produce the same or equivalent scenarios to the
14 generic scenarios presented in Figs. 5 and 6. It is also noted that, as with most
15 programming constructs, there are other similar ways of achieving the same or
16 similar functionality of an interface which may not be described herein, but
17 nonetheless are represented by the spirit and scope of the invention, i.e., it is noted
18 that it is at least partly the functionality represented by, and the advantageous
19 results enabled by, an interface that underlie the value of an interface.

20 21 **Conclusion**

22 Although the invention has been described in language specific to structural
23 features and/or methodological acts, it is to be understood that the invention
24 defined in the appended claims is not necessarily limited to the specific features or
25

1 acts described. Rather, the specific features and acts are disclosed as exemplary
2 forms of implementing the claimed invention.
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25